Everyone-Friendly Graphs

SIMPLE TWEAKS TO MATPLOTLIB GRAPHS FOR THE COLOR- AND NOT-SO-COLORBLIND

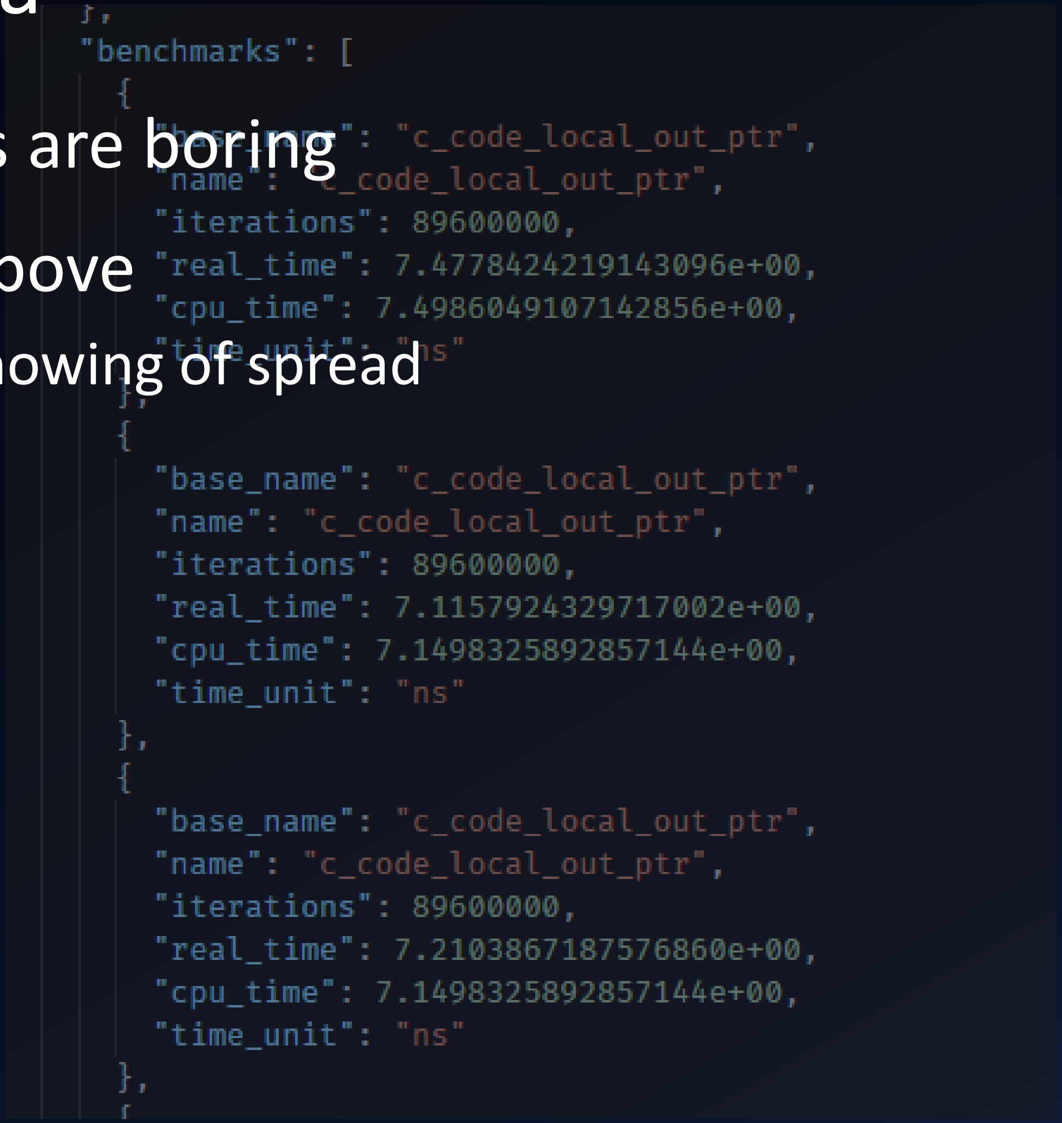
@thephantomderp

https://github.com/ThePhD/

NERD - December 2018, Boston Python

Statistical Benchmark Data

- Mean/Median/Mode printouts are boring
- Want visual indication of the above
  - plus standard deviation, visible showing of spread



```
},
"benchmarks": [
  {
    "base_name": "c_code_local_out_ptr",
    "name": "c_code_local_out_ptr",
    "iterations": 89600000,
    "real_time": 7.4778424219143096e+00,
    "cpu_time": 7.4986049107142856e+00,
    "time_unit": "ns"
  },
  {
    "base_name": "c_code_local_out_ptr",
    "name": "c_code_local_out_ptr",
    "iterations": 89600000,
    "real_time": 7.1157924329717002e+00,
    "cpu_time": 7.1498325892857144e+00,
    "time_unit": "ns"
  },
  {
    "base_name": "c_code_local_out_ptr",
    "name": "c_code_local_out_ptr",
    "iterations": 89600000,
    "real_time": 7.2103867187576860e+00,
    "cpu_time": 7.1498325892857144e+00,
    "time_unit": "ns"
  },
  {
```

Need Pretty Graphs!

- Quick solution desired
  - Fast to iterate
  - Do not want to put things in spreadsheets and do the excel thing

- Chose matplotlib + python to output my graphs

Step 0 – get files from places

- Use *argparse* library to handle command arguments
  - Data in JSON or CSV (but mostly JSON, so CSV is actually not implemented)

- Basic argparse stuff so we can get a file from the command line

  -
    ```
    import argparse

    parser = argparse.ArgumentParser(description=‘…’)
    parser.add_argument('-i', '--input', nargs='?’,
        default=‘blah.json’,
        type=argparse.FileType('r’))
    args = parser.parse_args()
    ```

Step 1 – load JSON, parse JSON

- Easiest part of the project
  - Turn JSON into a dictionary of name -> bucket of benchmark values
  - Store buckets in overarching categories, then store values based on entries
  - all_bars = benchmarks[“top level name”]
  - single_bar = all_bars[“single bar name”]
  - single_bar[“stats”], single_bar[“name”]

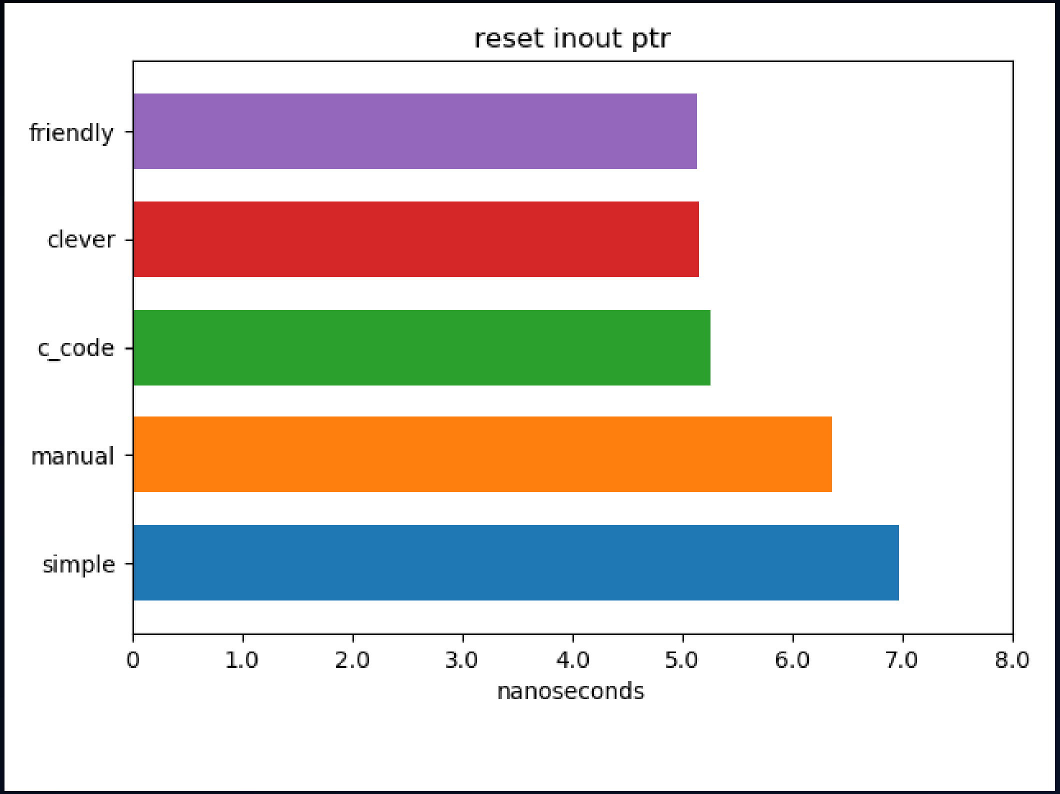
-
  ```
  import json

  j = json.load(args.input)
  benchmarks = parse_json(j, …)
  ```

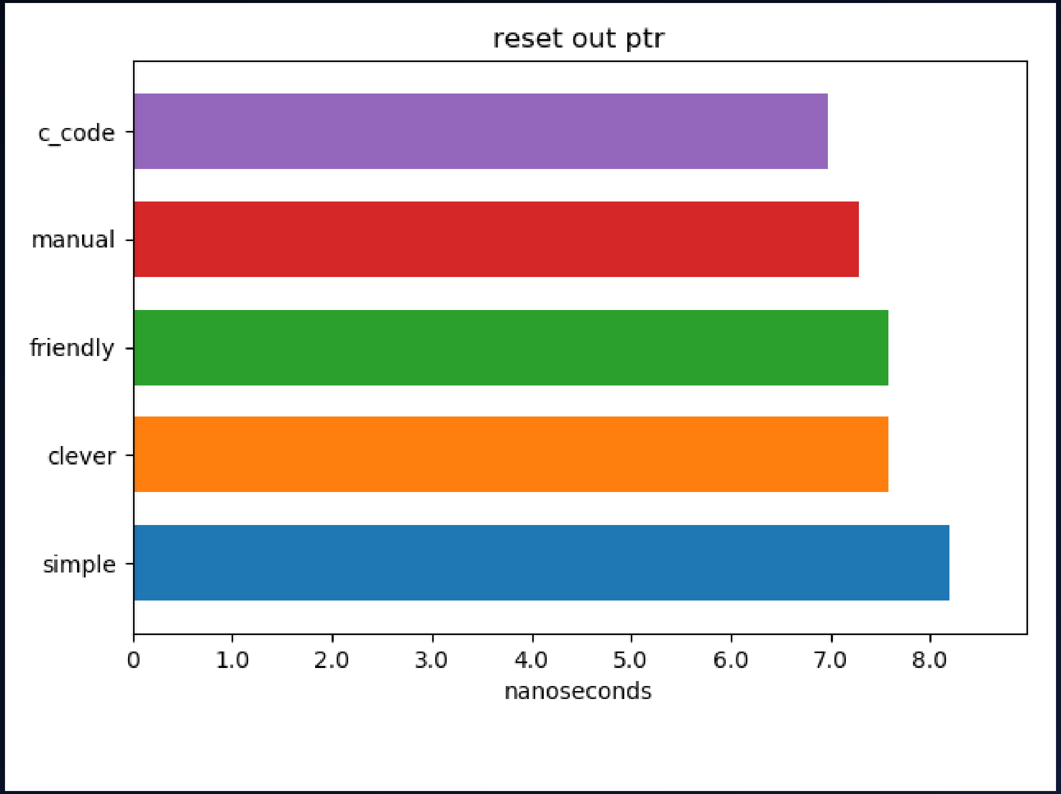
Step 2 – start using matplotlib

-
  ```
  import matplotlib
  import matplotlib.pyplot as plt

  # …

  def draw_graph(name, category, benchmarks, …):
        figures, axes = plt.subplots()
        for bi, benchmark in enumerate(benchmarks):
              # calculate bar_y, bar_height
              mean = benchmark["stats"]["mean"]
              axes.barh(bar_y, mean,
                        height=bar_height, align='edge’)

        axes.set_title(name)
        figures.tight_layout()
        figures.subplots_adjust(bottom=0.2)
  ```

Graphs Mk. 0

- Simple bar graph with basic labeling
  - … Wait a second, colors and names…?





Improvements

DESPERATELY NEEDS THEM

Improvements: Lounge<C++>

- How many values?
  - Show error bars
- Standard Deviation?
  - Show scatter of original values (superimposed? Maybe use transparency?)
- Is lower or higher better?
  - Order graph by desired metric, make clear in axis

Color array: keep stable color names

- Sort the benchmarks by bar name
  - Tag with incrementing integer id `color_index`, use to index into below array
  - Color stability between runs and between different graphs

-
  ```
  # some color constants, to help us be pretty
  # yapf: disable
  data_point_colors = [
       '#a6cee3',
       '#f255bb',
       ...
  ]
  ```

Apply `color` / `edgecolor`

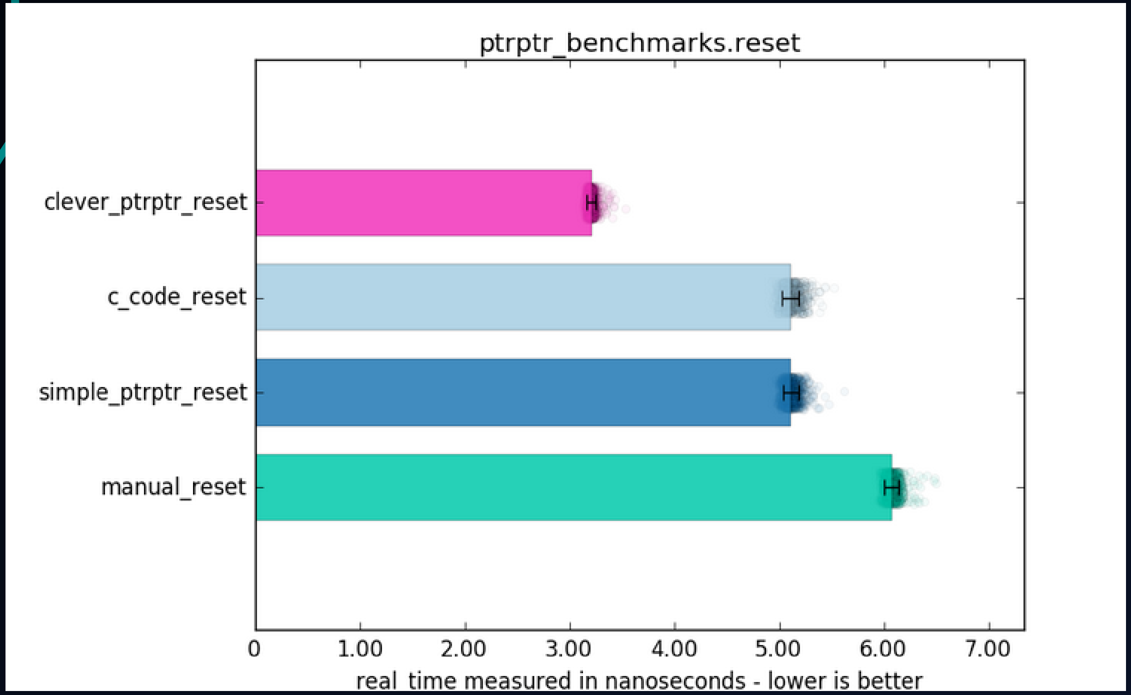
-
  ```
  color_index = benchmark["color_index"]
  color = data_point_colors[color_index]
  edgecolor = ‘#000000’

  axes.barh(bar_y, mean,
      height=bar_height,
      xerr=stddev, linewidth=0.2,
      edgecolor=edgecolor, color=color,
      hatch=hatch, align='edge’)
  ```

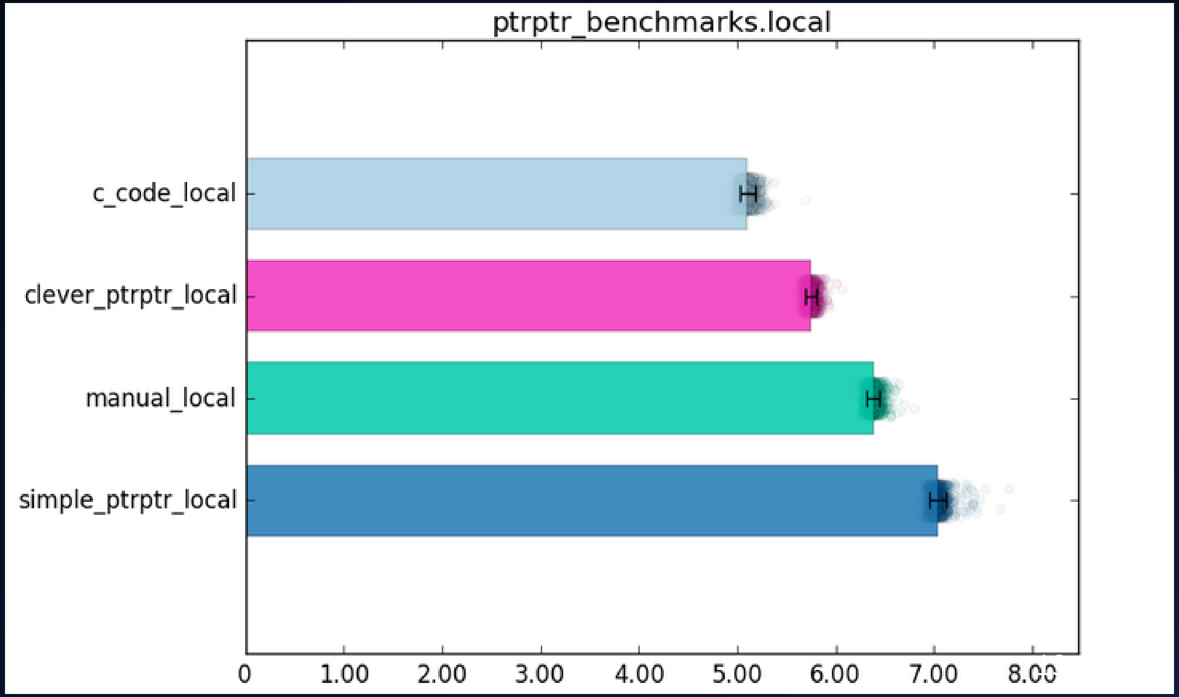
Apply `scatter` with Transparency

-
  ```
  xscatter = benchmark["data"]
  xlen = len(xscatter)
  yscatter = [
        bar_y + random.uniform(…)
        for _ in xscatter
  ]
  scatter_alpha = 0.20 if xlen < 11 else …

  scatter = axes.scatter(xscatter, yscatter,
        color=color, edgecolor=‘#000000’,
        linewidth=0.5, alpha=scatter_alpha)
  ```

Graph Mk. I

- Easier to read!
  - Value spread + error bars, colors for specific data points are sticky
  - All done! … Right?



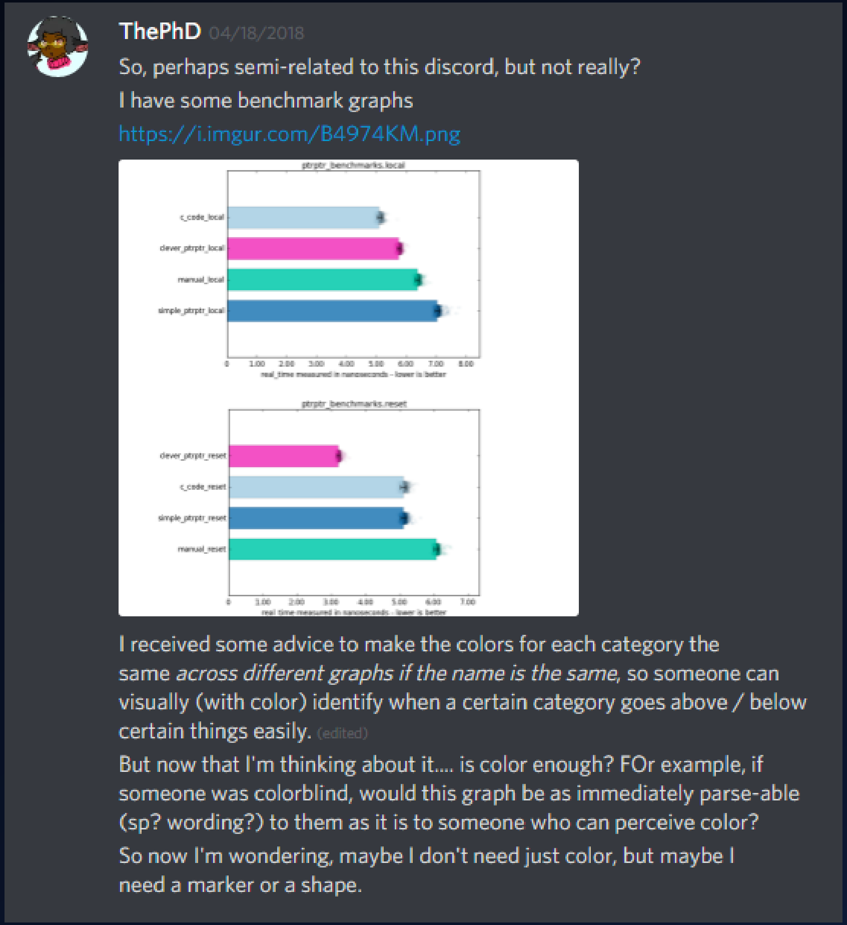


Right…?

- Met someone in #include discord
  - includecpp.org/

#include <C++>

DISCORD

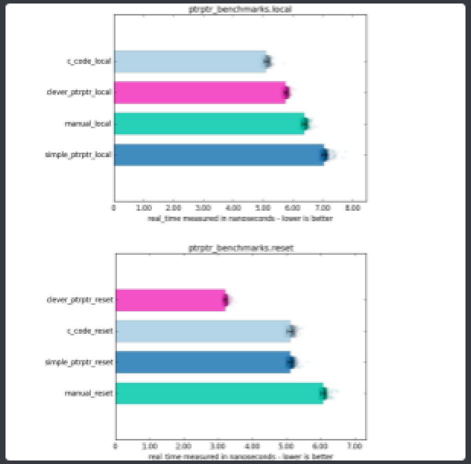


ThePhD 04/18/2018

So, perhaps semi-related to this discord, but not really?

I have some benchmark graphs

https://i.imgur.com/B4974KM.png



I received some advice to make the colors for each category the same *across different graphs if the name is the same*, so someone can visually (with color) identify when a certain category goes above / below certain things easily. (edited)

But now that I'm thinking about it…. is color enough? FOr example, if someone was colorblind, would this graph be as immediately parse-able (sp? wording?) to them as it is to someone who can perceive color?

So now I'm wondering, maybe I don't need just color, but maybe I need a marker or a shape.

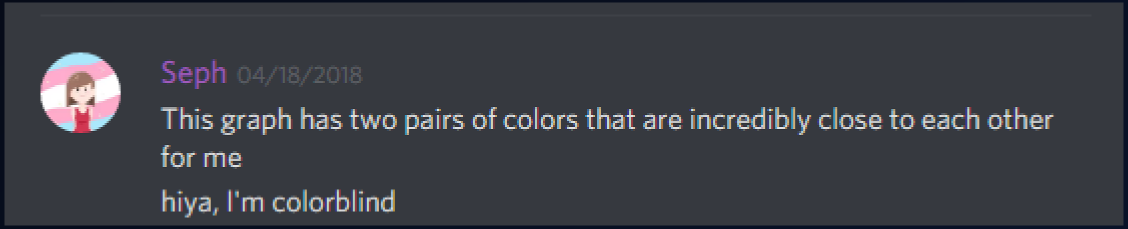
Wrong.

MAKING IT BETTER

More to do!

#include <C++>

- Not Colorblind friendly in the slightest!
- Seph started helping me, then Fred Tinguad, Olafur W., and Softwarebear…


Seph 04/18/2018
This graph has two pairs of colors that are incredibly close to each other for me
hiya, I'm colorblind

More Improvements!

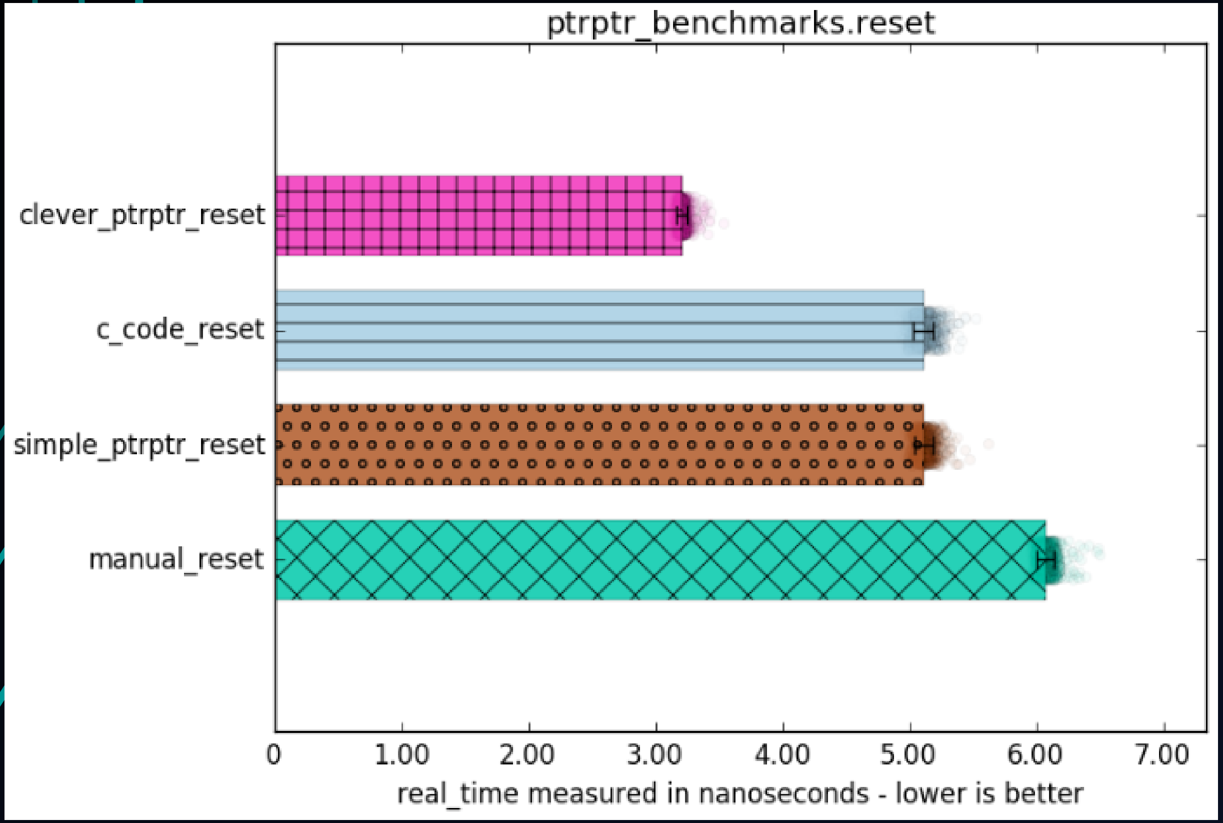
- Change Colors
  - More differentiation
  - Shapes/Patterns

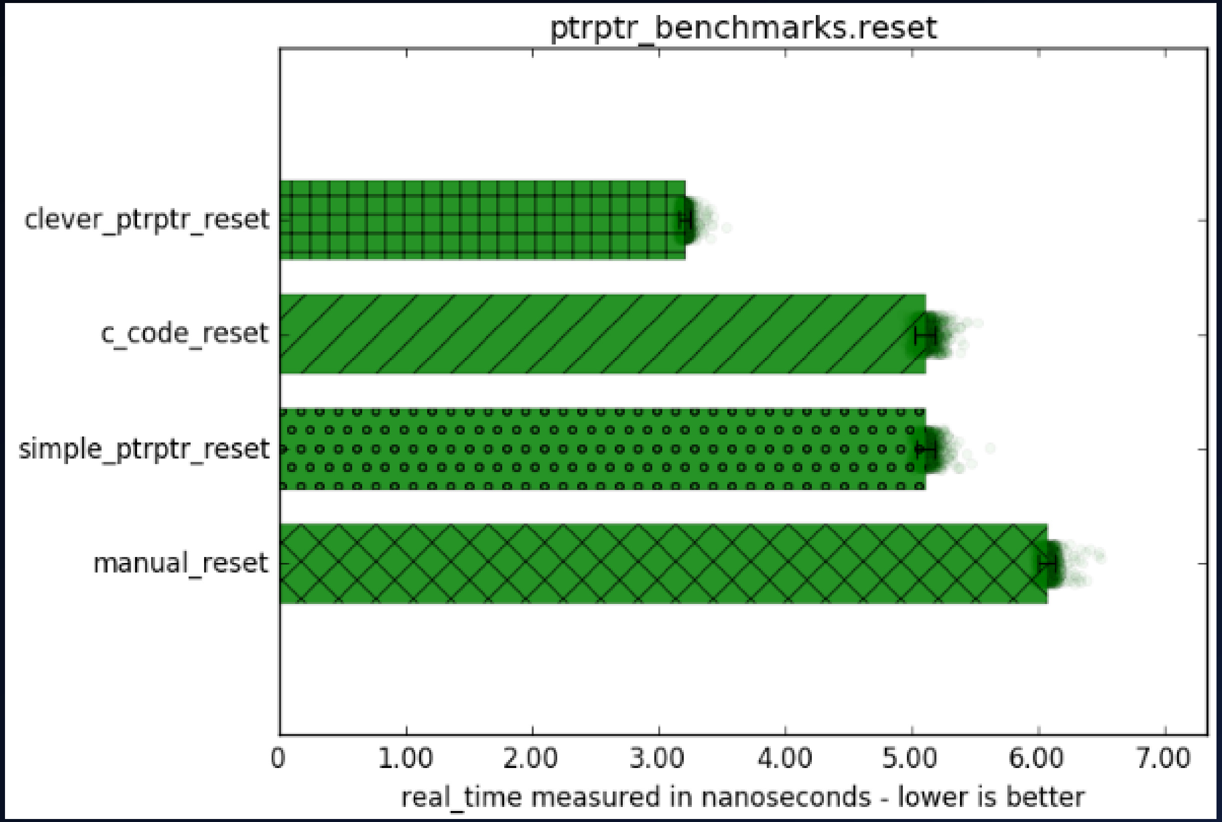
-
  ```
  # some pattern/color constants, to help us be pretty
  # yapf: disable
  data_point_aesthetics = [
       ('#a6cee3', '/’),
       ('#f255bb', ‘O’),
       ...
  ]
  ```

Use `edgecolor` plus `hatch` pattern

-
  ```
  axes.barh(bar_y, mean,
        height=bar_height,
        xerr=stddev, linewidth=0.2,
        edgecolor=edgecolor, color=color,
        hatch=hatch, align='edge’,
        error_kw={
              "capsize": 5.0, "mew": 1.2,
              "ecolor": 'black’,
        },
        alpha=0.82)
  ```

Still not quite right...



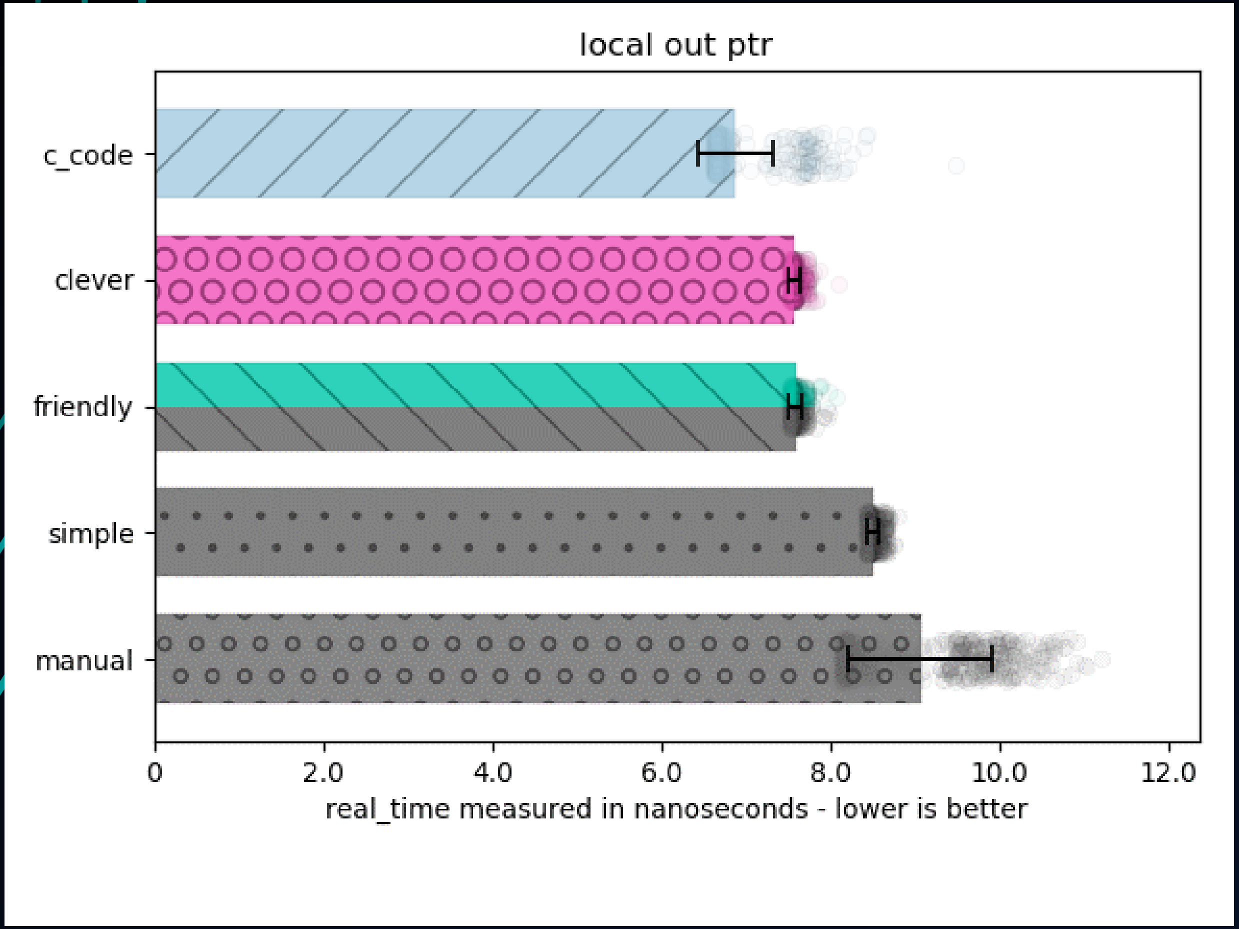


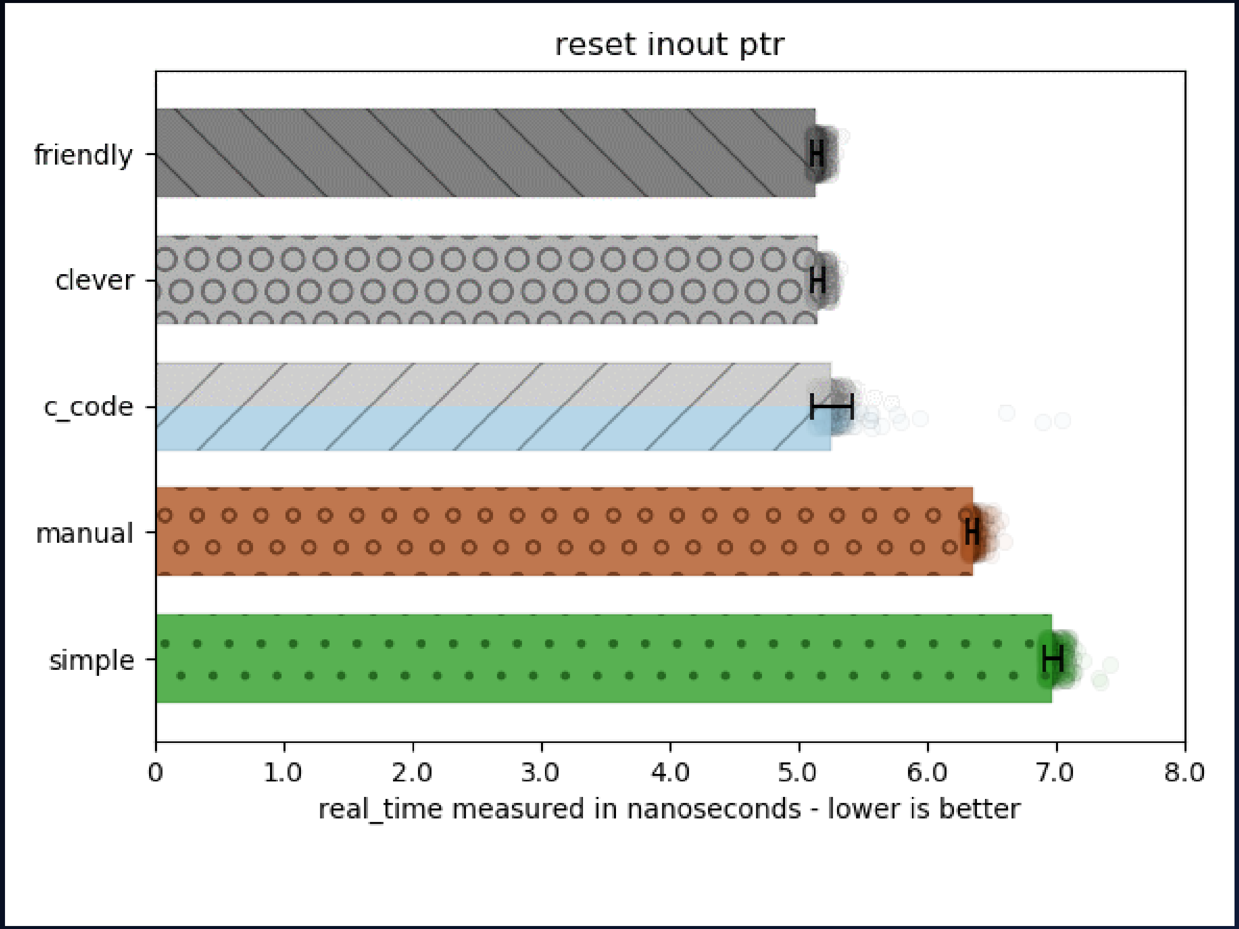
HSV to the Rescue

- Default shapes too dark / harsh
  - darken the RGB colors, but not too hard
  - Convert Red Green Blue (RGB) to Hue Saturation Value (HSV)
  - Lower the V in HSV (also known as “Lightness”)

-
```
color = …
colorhsv = matplotlib.colors.rgb_to_hsv(
        matplotlib.colors.hex2color(color)) # 35FF6A -> (53,255,106)
colorhsv[2] *= 0.6 # decrease V value
edgecolor = matplotlib.colors.hsv_to_rgb(colorhsv)
```

Beautiful. For Everyone.





Colorblind Friendly = *Everyone* Friendly

- It started as being just a Colorblind investigation…
- Accommodating disability brings gains beyond just feel-good cred
- "I want to improve my bottom line"
  - Ask someone with greater challenges than yourself (colorblind, broken arm)
  - How they like it / handle it often makes it easier for the able-bodied too!

@thephantomderp

https://www.patreon.com/thephd

https://www.linkedin.com/in/thephd

https://github.com/ThePhD/

Thank You

FOR LISTENING!